EE 341 – Lab 3

Frequency Analysis using FFT

Lab Objectives

The purpose of this lab is to get you familiarized with using the Fast Fourier Transform (FFT) to study the frequency content of discrete-time signals. You will firstly inspect and describe the frequency contents of some generated signals and provided audio signals. Then, you will examine how several transformations on these signals affect the frequency content of them.

Task 1: Using the FFT function

In this task you will learn how to use the `fft` function provided by the SciPy package. Firstly, to use `fft`, you need to import the `scipy.fftpack` package as follows.

```
from scipy.fftpack import *
```

Put it after your other import statements. In this lab you will also need `numpy`, `pyplot`, and `simpleaudio`. Refer to the previous labs on how to import or use them.

Next, refer to the reference document (here) of the `fft` function to have a general overview of the parameters and usages of the function. The `fft` function computes the Discrete Fourier Transform (DFT) of an array. The result of `fft` is an array of complex numbers so you will need to look at the magnitude and phase of them separately. The functions `np.abs` and `np.angle` are useful for obtaining the magnitude and phase of a complex number or a complex-valued array. Also, since the FFT only has values at discrete frequencies, it may be useful to do the plots with `plt.stem` to reinforce the idea, but continuous frequency plots (i.e. using `plt.plot`) are often used since they are closer to the DTFT that you are ultimately interested in.

The FFT outputs an array that the indices correspond to the range $0 \leq \omega < 2\pi$ (or $0 \leq f < 1$). You are probably more familiar with seeing the spectrum plotted over the range $-\pi \leq \omega < \pi$ (or $-0.5 \leq f < 0.5$). The `fftshift` function can be used for this purpose.

Assignment

Plot the magnitude of the FFT of the following signal before and after the `fftshift`:

$$x[n] = 1 + cos(2\pi fn)\,; 0 \leq n < 128$$

for the cases where $f = 0.25$ and $f = 0.5$.

In your notebook, include the 3-part plot (using `plt.stem`) for each signal: unshifted DFT, shifted DFT and the shifted DFT with Hertz frequency assuming a sampling rate of 10 kHz. Discuss why the frequency peak locations make sense.

Task 2: Frequency Shifting

Recall that multiplying by a complex exponential in time (or a cosine, which is comprised of a pair of complex exponentials) results in a frequency shift. For each of the following sequences, let $f_1 = 0.15$ and $0 \leq n < 256$.

Developed/revised by Yibo Cao, Kamil Jiwa, and Tai Chen 2019

To evaluate sinc, use the `np.sinc` function provided by NumPy. Plot the magnitude and phase plots (using `plt.plot`), where the magnitude and phase plots are over the range $-0.5 \leq f < 0.5$ (normalized frequency), i.e. use `fftshift`.

(a) $x_1[n] = \text{sinc}\big(f_1 \cdot (n - 32)\big)$
(b) $x_2[n] = \text{sinc}\big(f_1 \cdot (n - 32)\big) \cdot (-1)^n$
(c) $x_3[n] = \text{sinc}\big(f_1 \cdot (n - 32)\big) \cdot cos(2\pi f_2 n)$ where $f_2 = 0.2$
(d) $x_4[n] = \text{sinc}\big(f_1 \cdot (n - 32)\big) \cdot cos(2\pi f_3 n)$ where $f_3 = 0.4$

Display the plots for (a) – (d). Treating the plots as frequency response of filters, state what type of filter each of them corresponds to (low pass, high pass, etc.). For (d), explain why (d) does not have a flat frequency response in the passband.

Task 3: Starting from Continuous Time Signals

Download two sounds files from the class website, picking one that you think will have more high-frequency content and one that will have more low-frequency content. Load each sound using the function provided in Appendix A. Record the sampling rate and length of the samples (in seconds). Play the sound using the following statement introduced in Lab 1:

```
sa.play_buffer((samples * 32767).astype('int16'), 1, 2, sr).wait_done()
```

where `samples` is the array of the audio samples and `sr` is the sampling rate.

Plot the time-domain waveform and the FFT of each sound. For the frequency plot, use your understanding of the relation between discrete and continuous time and knowledge of the sampling rate to scale the frequency axis to match the continuous time range in Hertz. Comment on whether the frequency content matches your expectation. Then, perform the following modifications to the signals.

(a) Modify your signals by multiplying the time-domain signal by either $(-1)^n$ as in Task 2 (b), or a cosine as in Task 2 (c). Play the sounds and plot the frequency content of the new sounds. What is the effect of the frequency shift on how they sound compared to the original?
(b) Modify your signals by time-scaling: $y[n] = x[2n]$, which results in frequency scaling. Play the sounds and plot the frequency content. How does frequency scaling compare to frequency shifting?
(c) One way to implement an ideal filter is to zero out frequency terms. Implement a high-pass filter with a cut-off of $f_c = 0.25$ by zeroing out the low frequency terms in the FFT of the sounds you chose. Then take the inverse FFT of the result using `ifft` and `ifftshift` to get the modified signal. Play the sounds and plot the frequency content of the new sounds. (You may need to scale the filtered sound if it is hard to hear, since you've eliminated a lot of the energy in it.) Discuss the impact of this operation on the sounds.

In your notebook, include the frequency plots for the original and modified versions of each of your sounds. Provide details of the modifications that you did in each part and explain the impact of these operations on what you hear.

Lab Submission Requirements

Submit your completed Jupyter Notebook in the `ipynb` format to Canvas. The notebook file can be obtained by clicking the menu: File – Download as – Notebook or directly accessed from the ee341lab folder in your home directory. Make sure to include the code of all the 3 tasks in code cells and include the required discussions of each task in text cells.

Appendix A: Loading WAV Files

To load sound data from WAV files into your Python program, firstly copy the files you want to load into the same directory where your Jupyter Notebook is stored (that is the ee341lab folder in your home directory if you followed our convention in Lab 1). Then, add the following import statement after your other imports:

```
import scipy.io.wavfile as wav
```

Ideally, you should be able to directly use `wav.read` from this package to load files. However, WAV files can have different quantization formats and number of channels that will cause troubles later on. Therefore, we provide you the following helper function to help you deal with different formats.

```
# Load a WAV file.
# Return the sampling rate and the sample array.
def wav_load(file_name):
  # Load the raw data.
  sr, data = wav.read(file_name)
  # Only use the first channel.
  if data.ndim > 1:
    data = data[:, 0]
  # Convert to float32 quantization.
  kind = data.dtype.kind
  bits = data.dtype.itemsize * 8
  data = data.astype('float32')
  if kind == 'i' or kind == 'u':
    data = data / 2 ** (bits - 1)
    if kind == 'u':
      data = data - 1
  return sr, data
```

Example usage:

```
sr, samples = wav_load("do.wav")
```

This will load the file `do.wav`, putting the sampling rate into the variable `sr` and putting the sample array into the variable `samples`.